
BOUNDED LAZY INITIALIZATION

**JACO GELDENHUYS
NAZARENO AGUIRRE
MARCELO F. FRIAS
WILLEM VISSER**



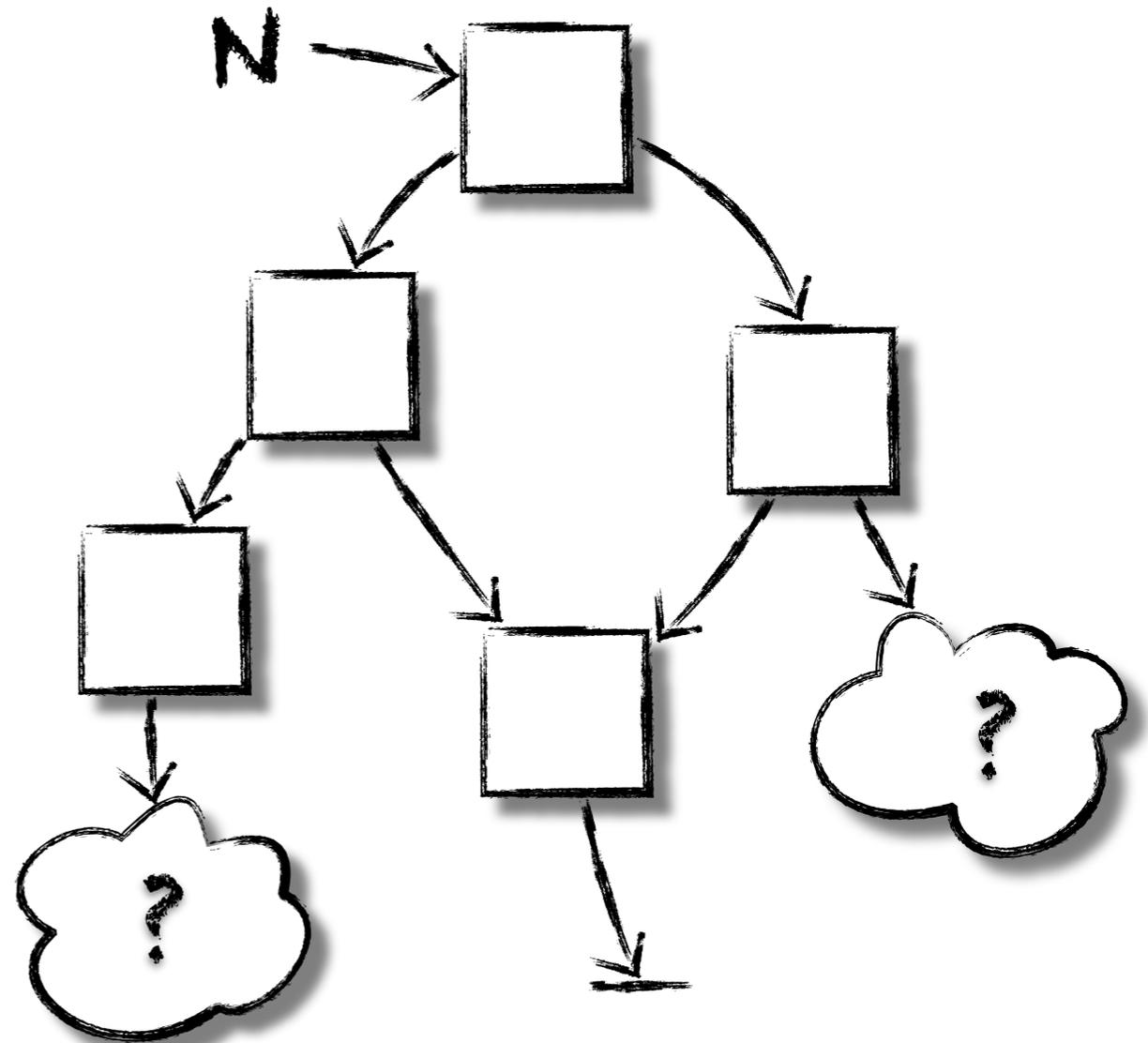
BOUNDED *LAZY* INITIALIZATION



MOTIVATION

Improve the ability of **symbolic execution** to investigate code that involves **dynamic structures** (linked lists, trees, graphs, etc.)

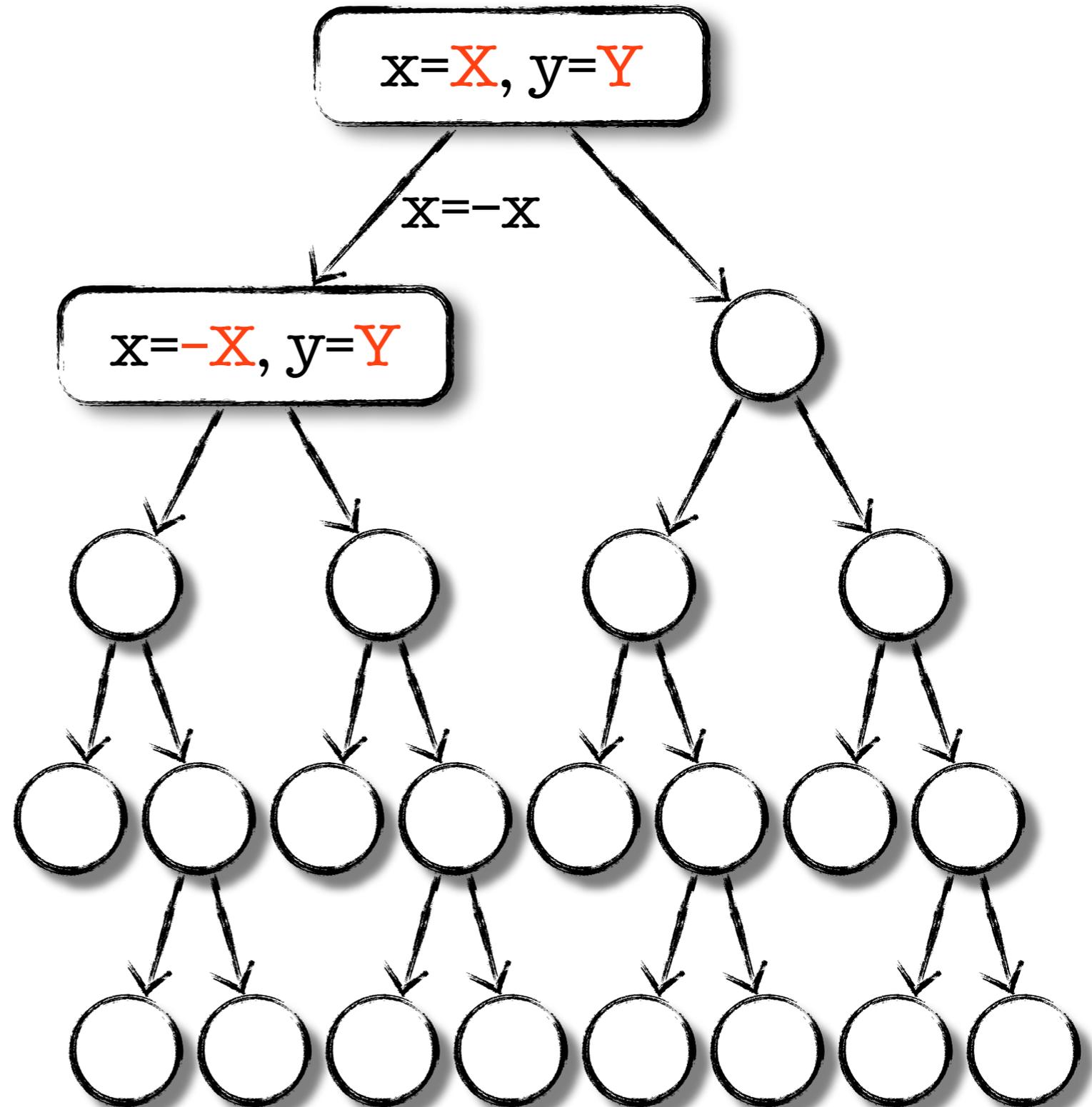
```
Node insert(Node N, Key k) {  
  if (N == null) {  
    N = new Node(k);  
  } else if (N.k < k) {  
    N.l = insert(N.l, k);  
  } else if (N.k > k) {  
    N.r = insert(N.r, k);  
  }  
  return N;  
}
```



SYMBOLIC EXECUTION

Symbolic execution explores all possible execution paths, using symbolic – not concrete – values for variables.

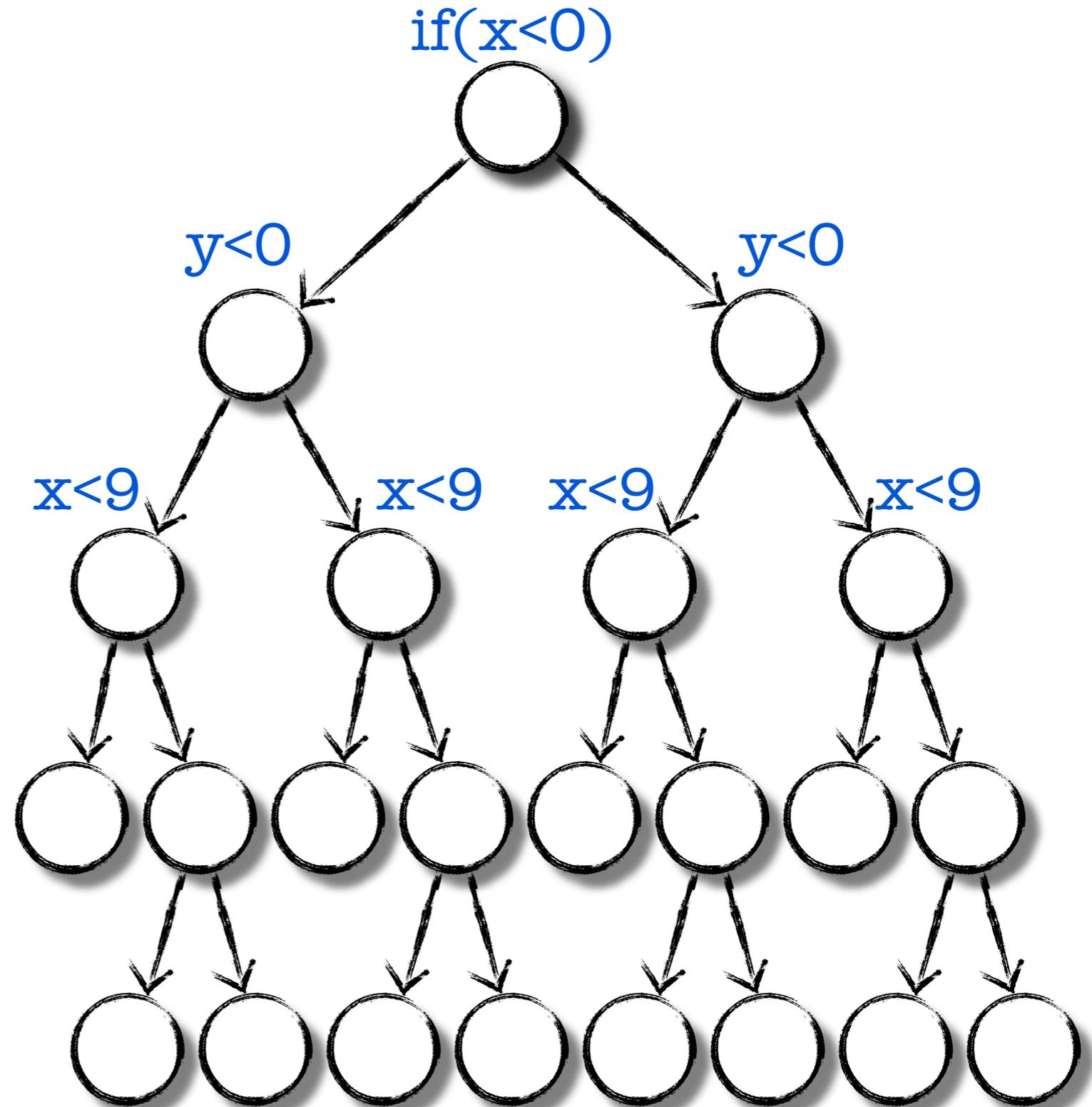
```
int m(int x, y) {  
  if (x<0) x=-x;  
  if (y<0) y=-y;  
  if (x<9) return 1;  
  if (8<y) return -1;  
  return 0;  
}
```



SYMBOLIC EXECUTION

Branching points in the execution tree corresponds to branching points in the programs.

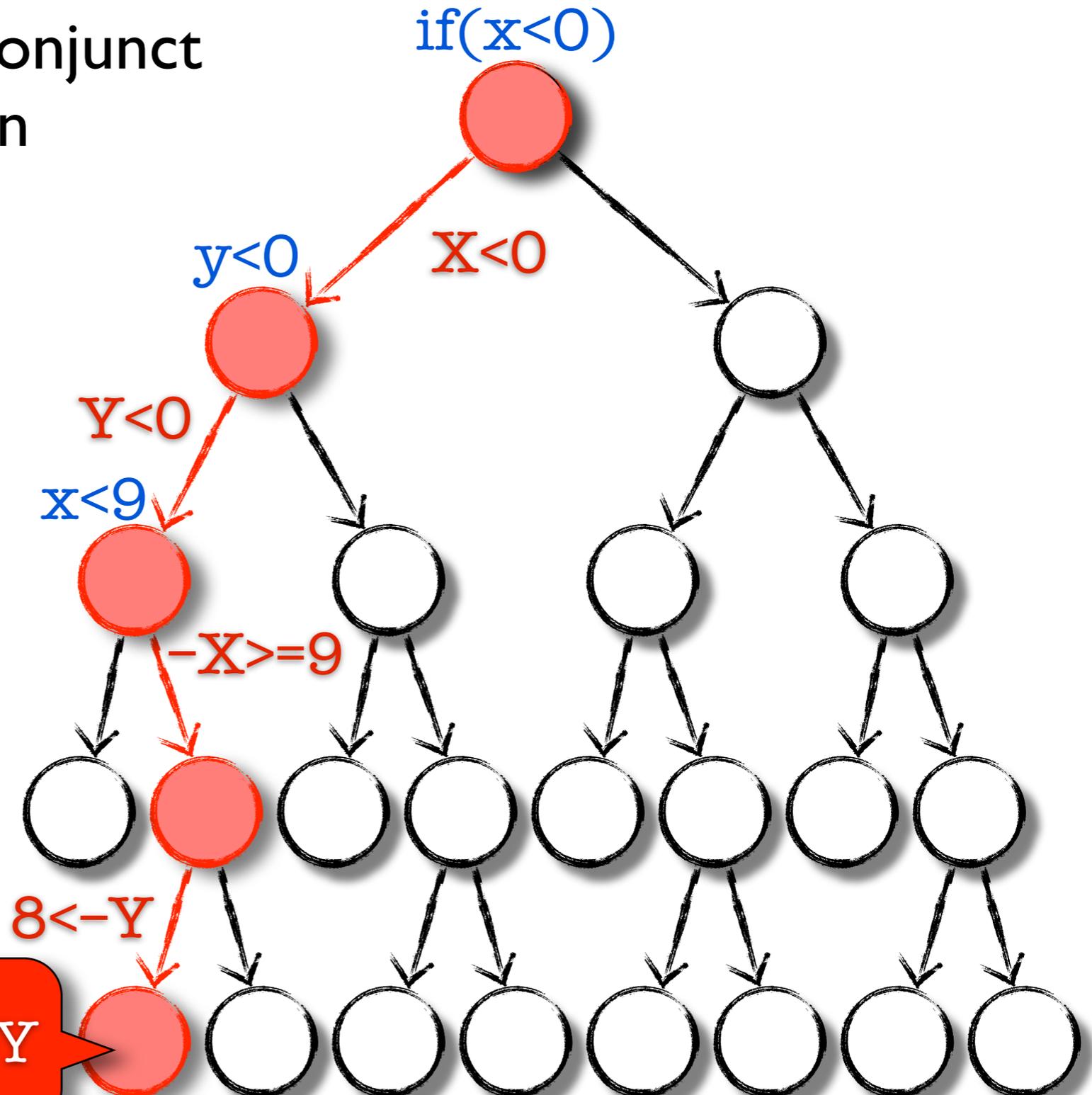
```
int m(int x, y) {  
    if (x<0) x=-x;  
    if (y<0) y=-y;  
    if (x<9) return 1;  
    if (8<y) return -1;  
    return 0;  
}
```



SYMBOLIC EXECUTION

Each edge contributes a conjunct to the **path condition** of an execution path.

```
int m(int x, y) {  
  if (x<0) x=-x;  
  if (y<0) y=-y;  
  if (x<9) return 1;  
  if (8<y) return -1;  
  return 0;  
}
```



PATH CONDITION

$X < 0 \wedge Y < 0 \wedge -X \geq 9 \wedge 8 < -Y$

LAZY INITIALIZATION

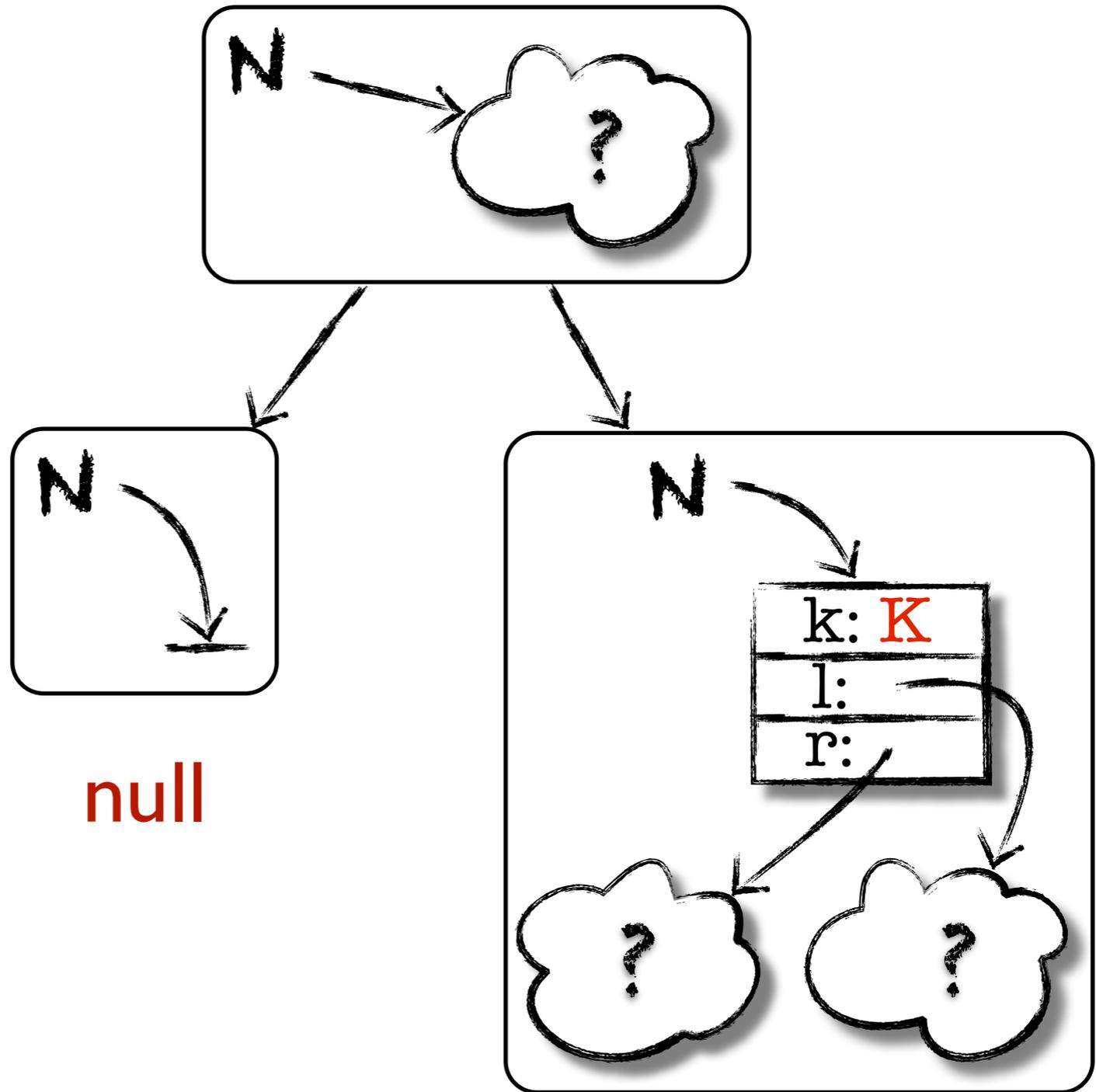
Dynamic structures are explored symbolically. All references remain symbolic until their values become relevant. At that point, they are initialized as

- null,
- a new node, and
- a reference to one of the existing nodes.

All execution paths from these possibilities are explored.

LAZY INITIALIZATION

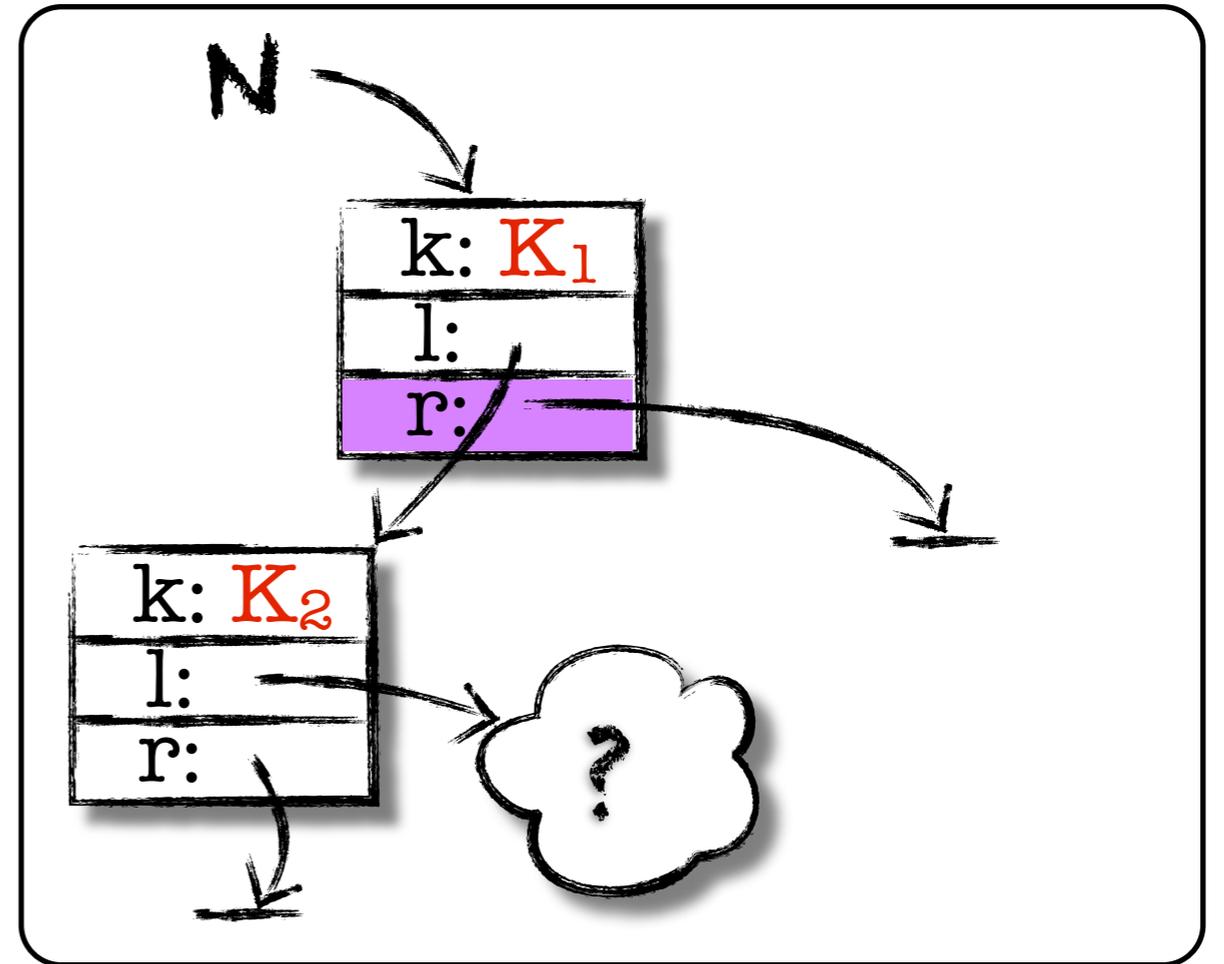
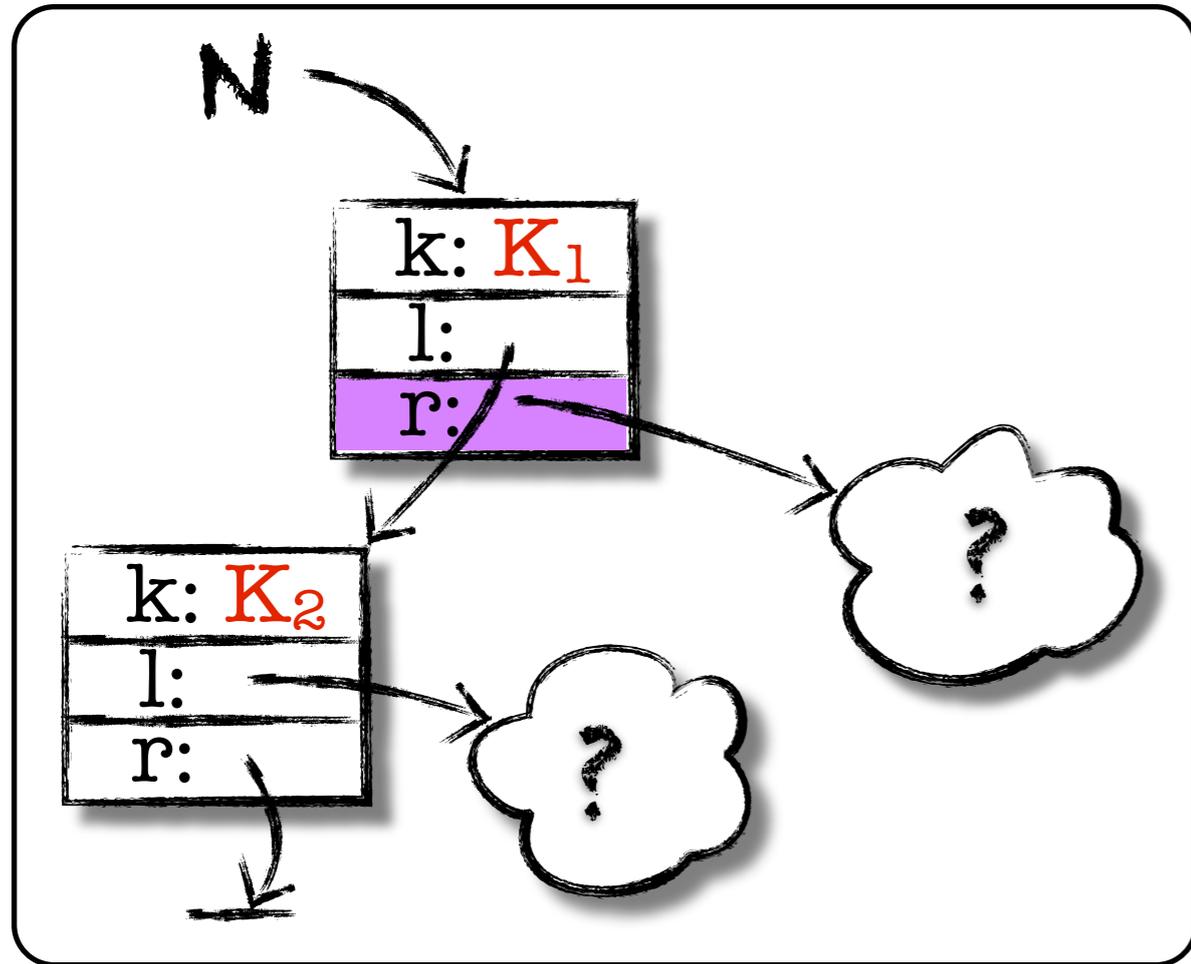
```
Node N;  
...  
if (N == null) {...}
```



null

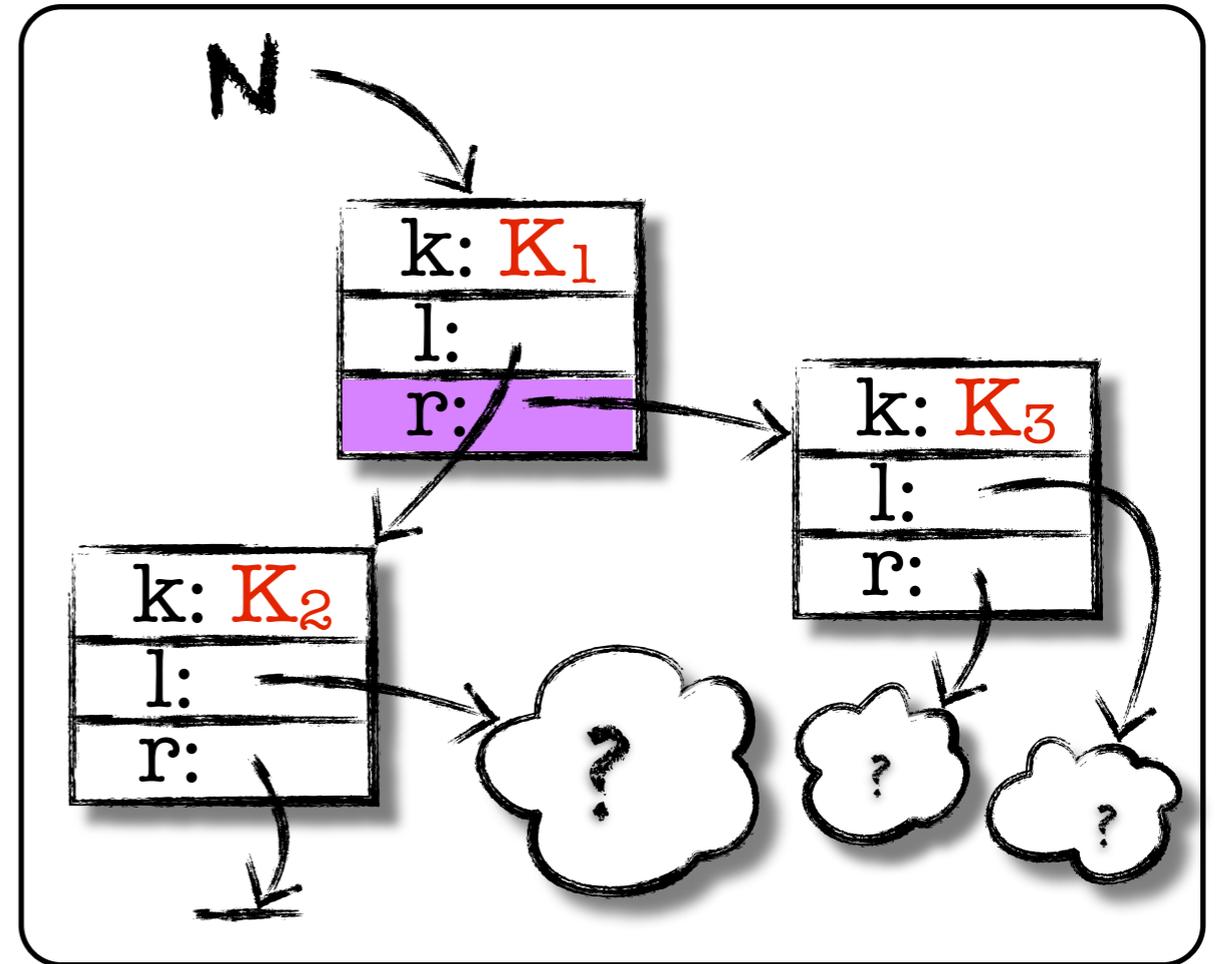
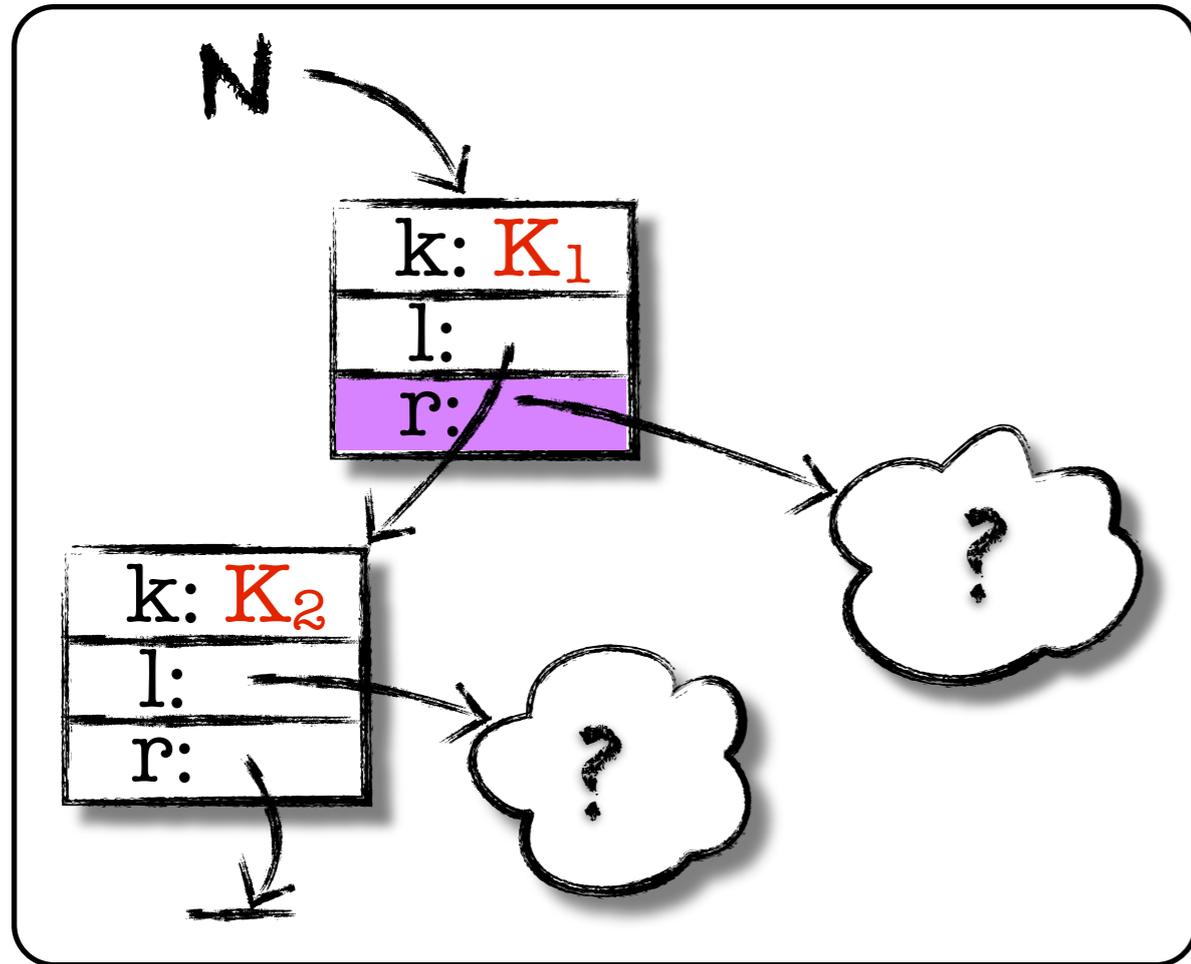
new node

LAZY INITIALIZATION



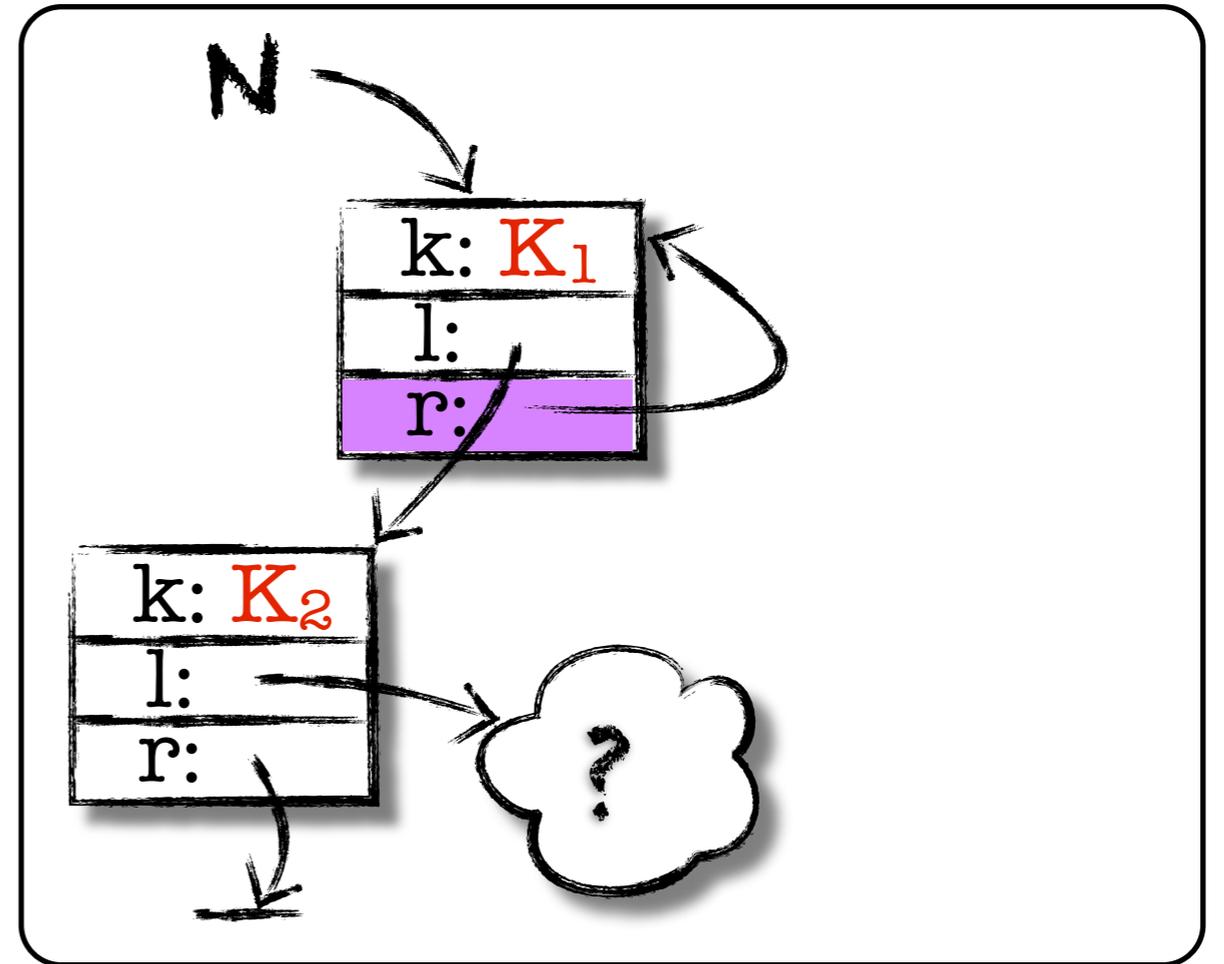
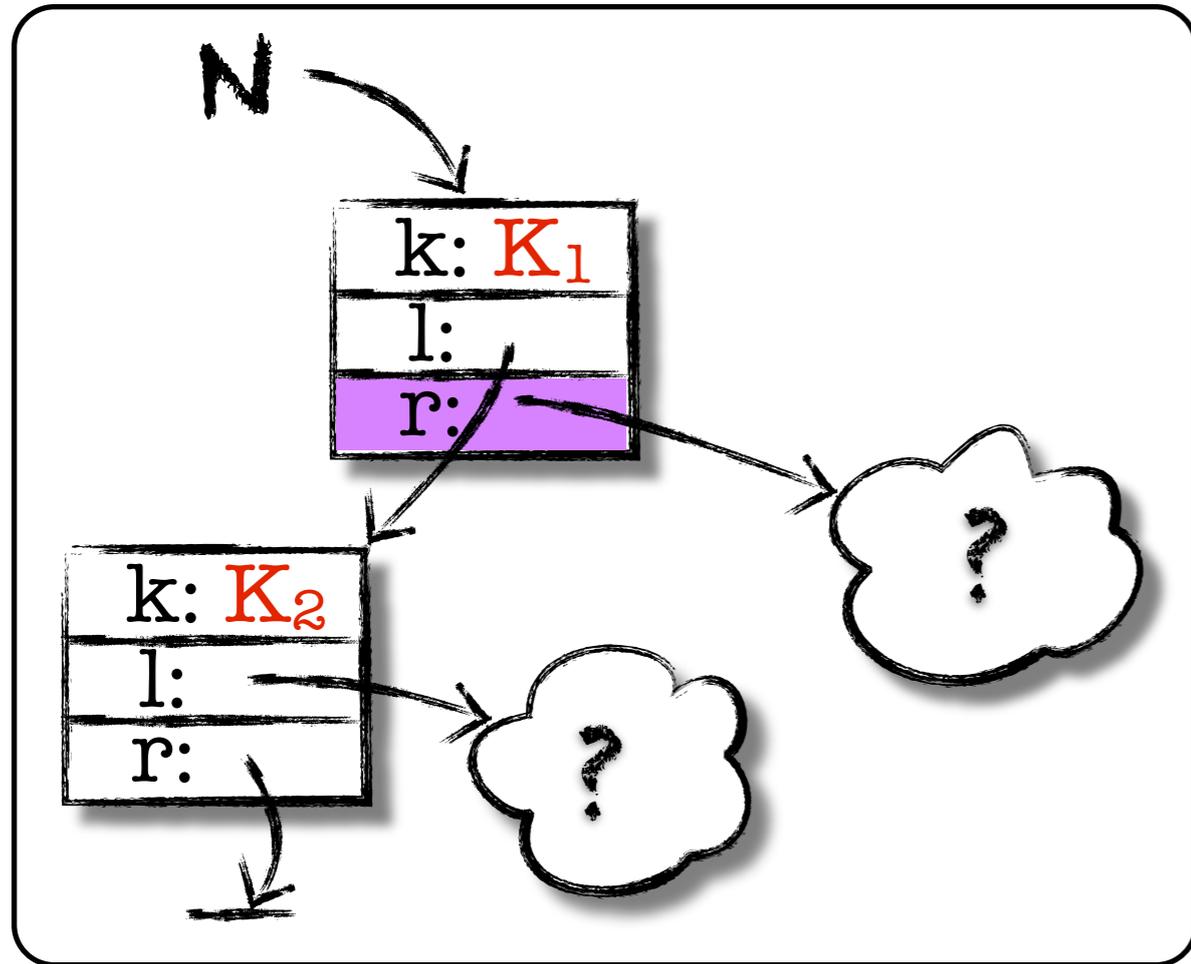
null

LAZY INITIALIZATION



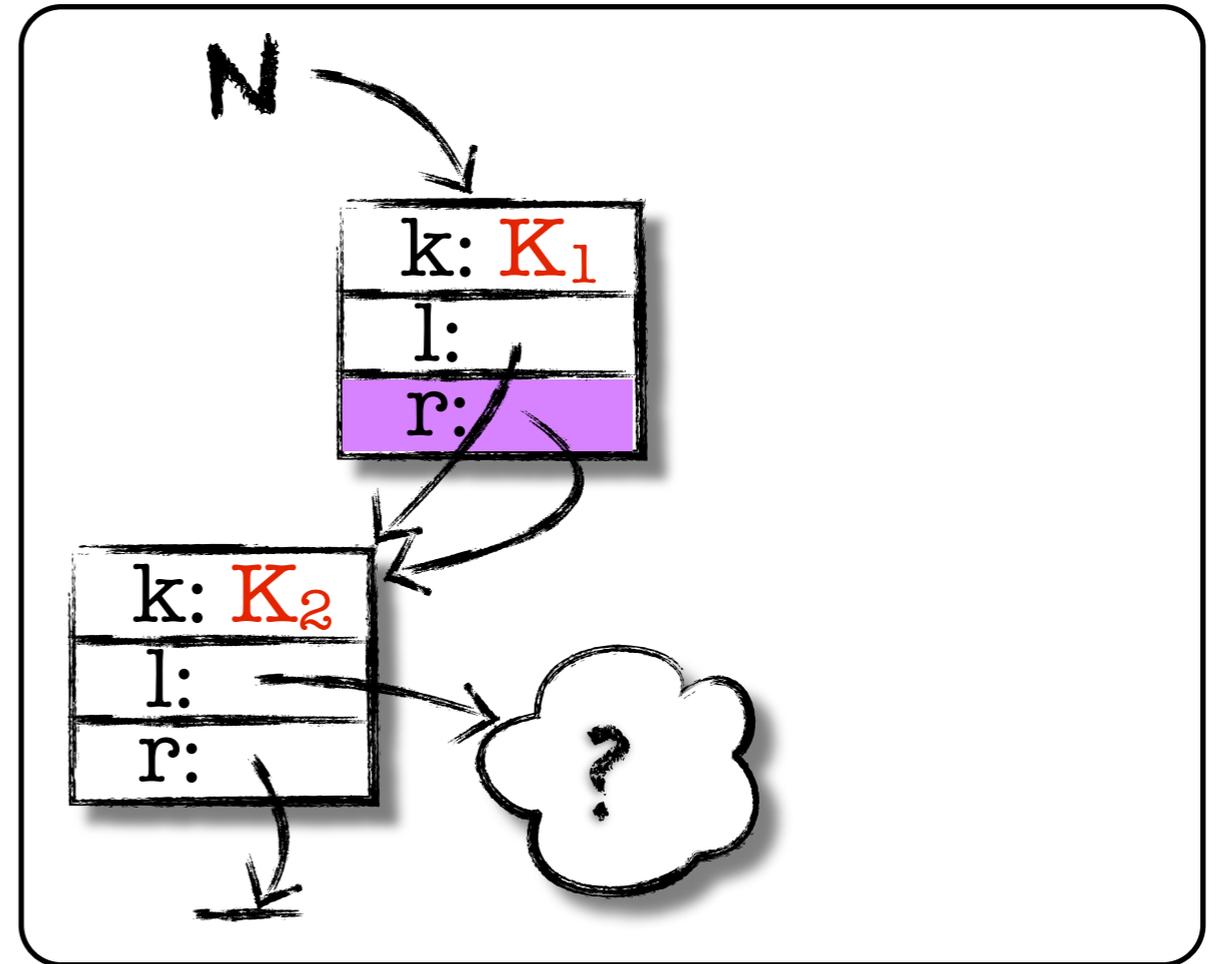
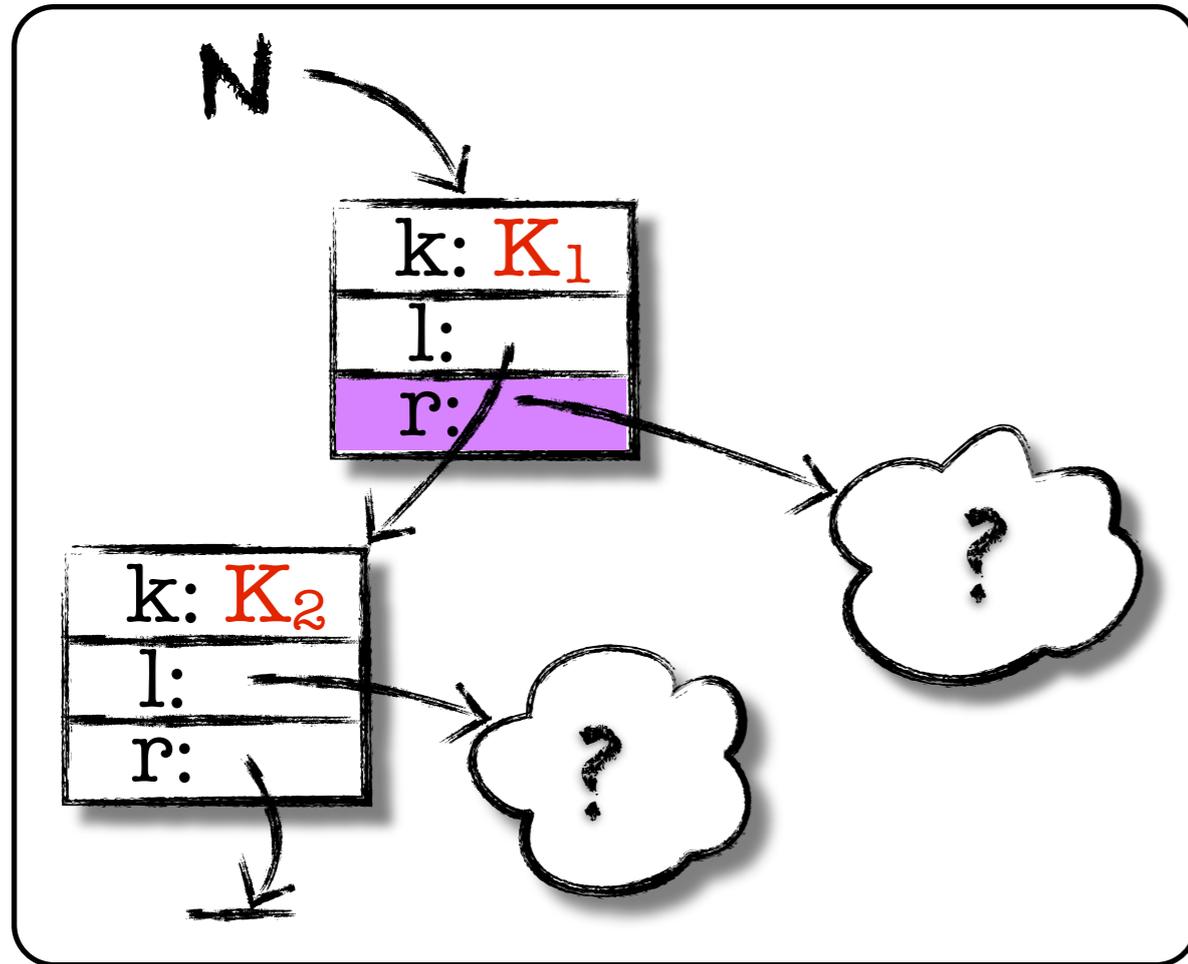
new node

LAZY INITIALIZATION



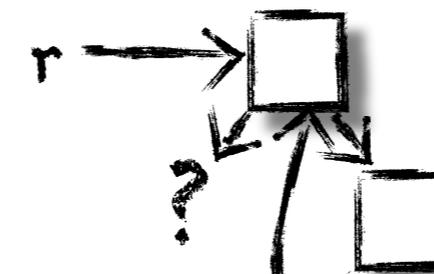
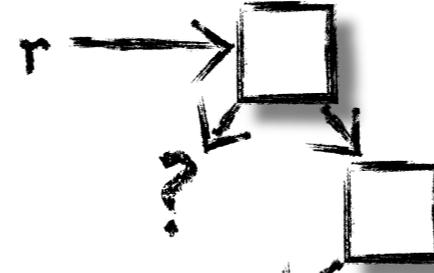
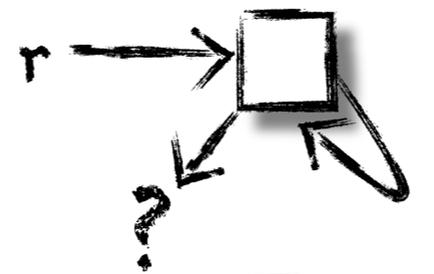
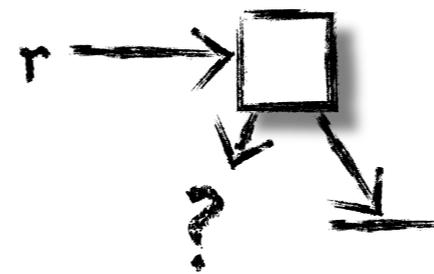
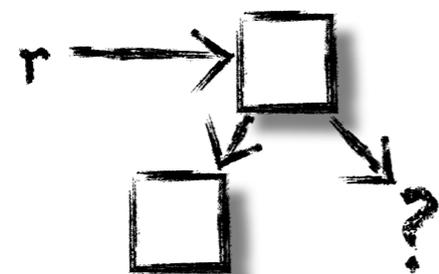
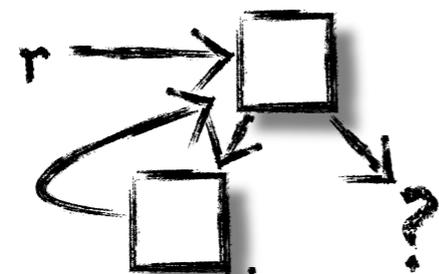
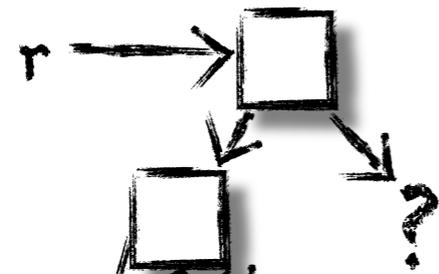
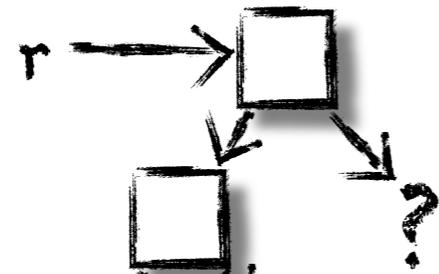
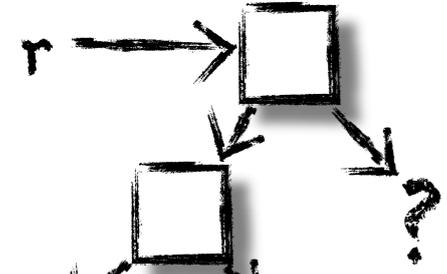
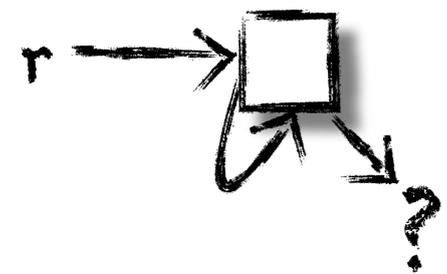
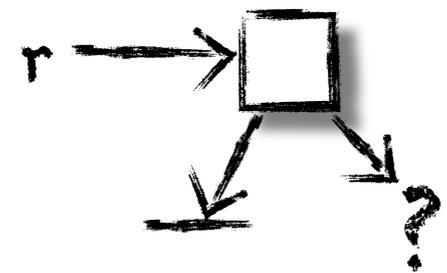
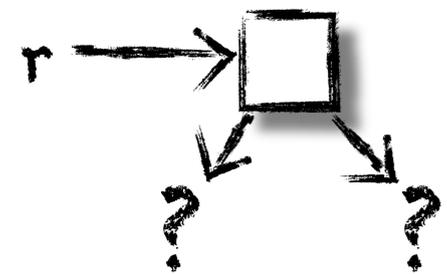
point back to root node

LAZY INITIALIZATION

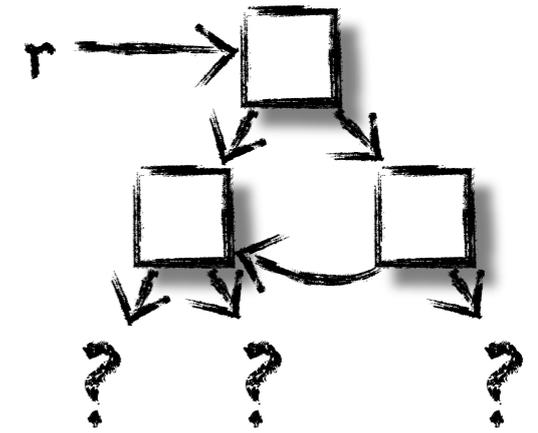
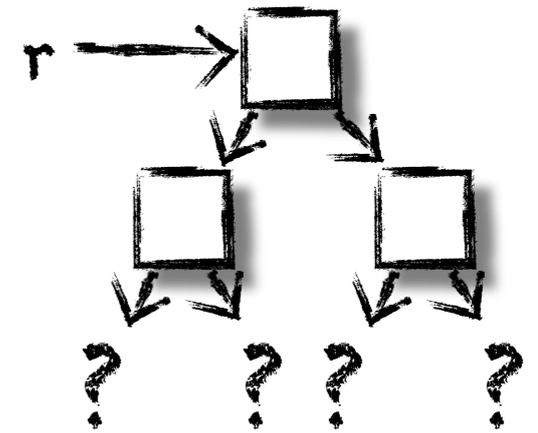


point to left child

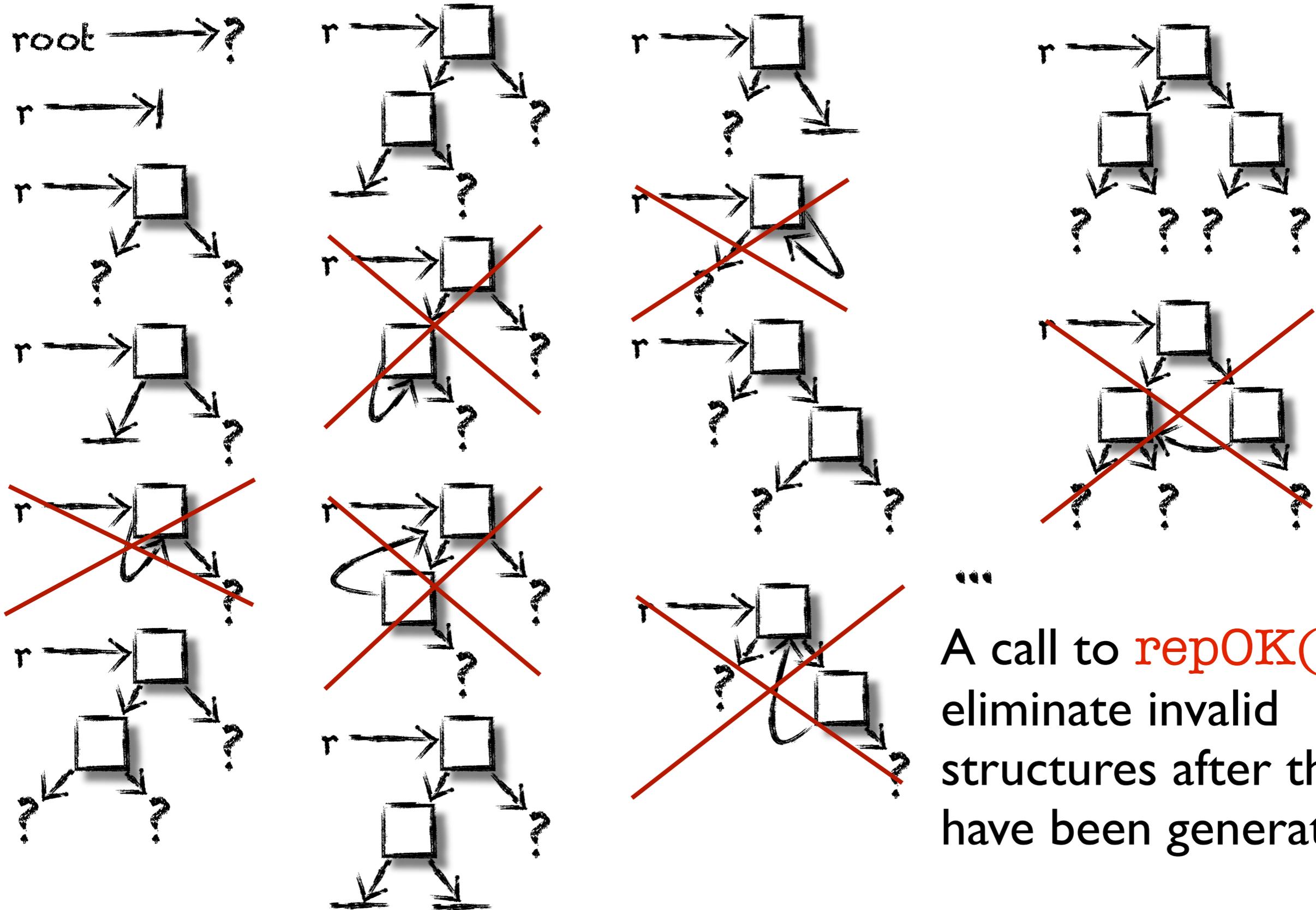
LAZY INITIALIZATION



...

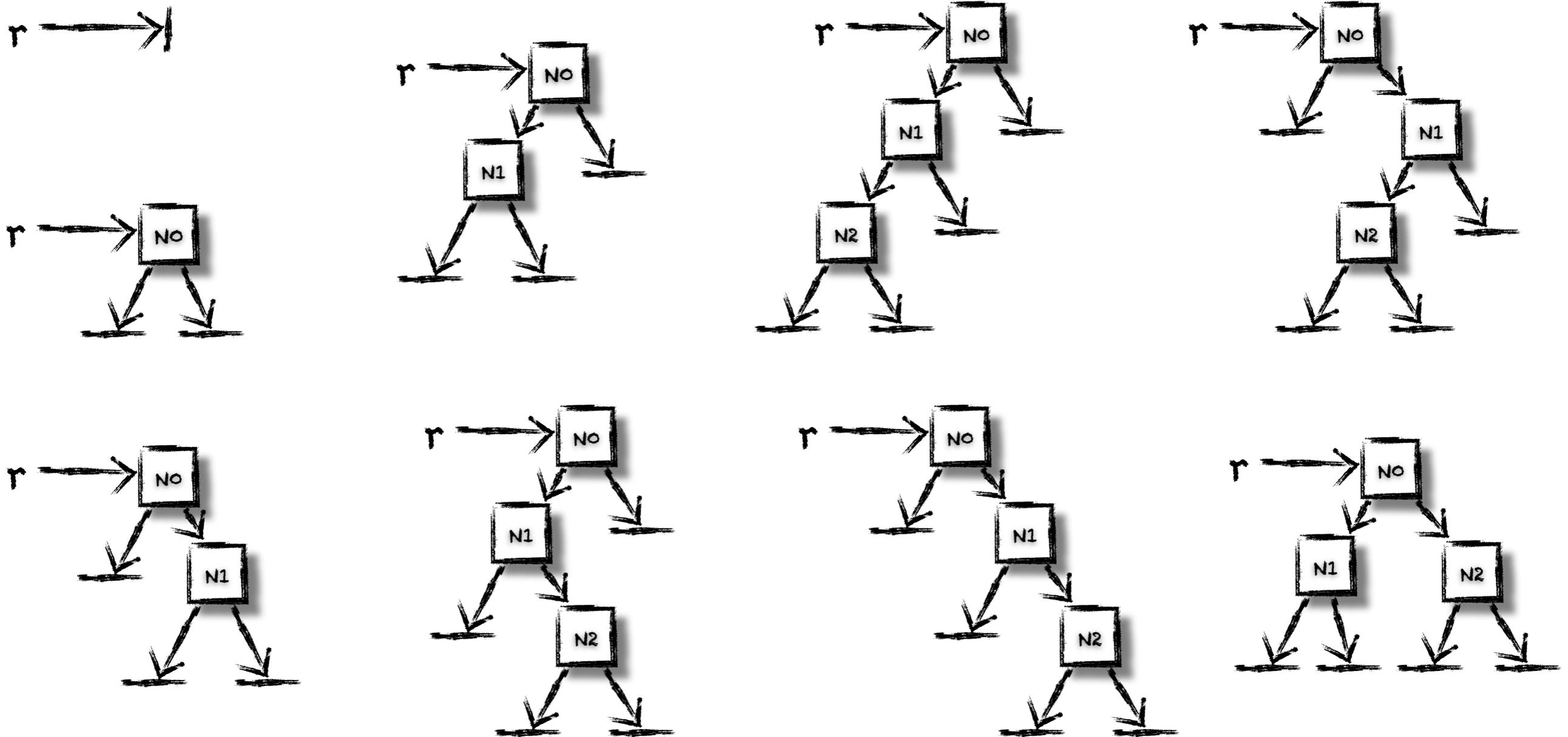


LAZY INITIALIZATION



“ A call to `repOK()` can eliminate invalid structures after they have been generated.

BOUNDED LAZY INITIALIZATION



Central idea: pre-generate all valid field values by solving the following SAT problem: is there a structure where the left (or right) field of N_x is N_y ?

LAZY INITIALIZATION

root	null, NO
NO.left	null, NO, N1
NO.right	null, NO, N1
N1.left	null, NO, N1, N2
N1.right	null, NO, N1, N2
N2.left	null, NO, N1, N2
N2.right	null, NO, N1, N2

BOUNDED LAZY INITIALZTION

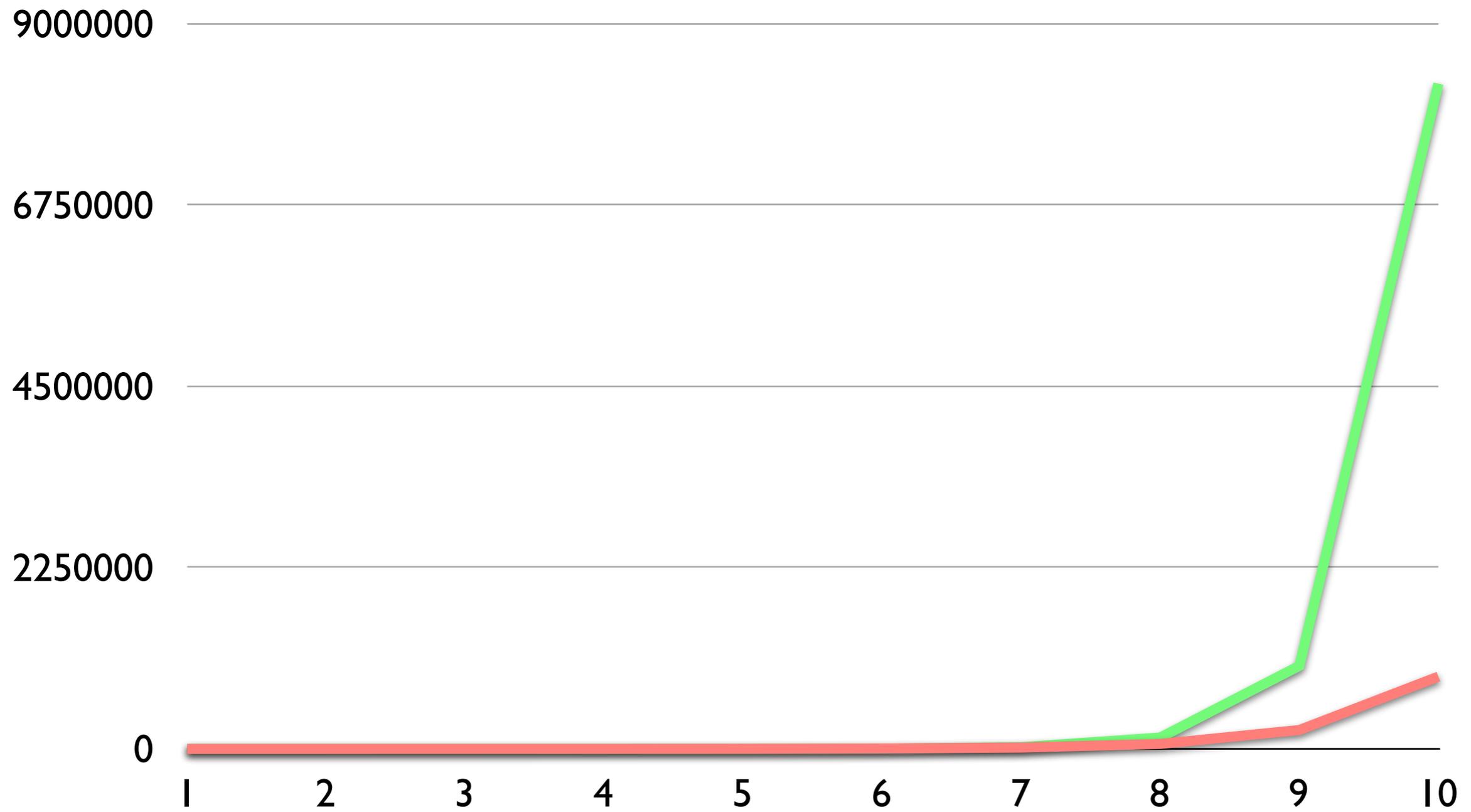
root	null, NO
NO.left	null, N1
NO.right	null, N1, N2
N1.left	null, N2
N1.right	null, N2
N2.left	null
N2.right	null

LAZY VS BOUNDED-LAZY

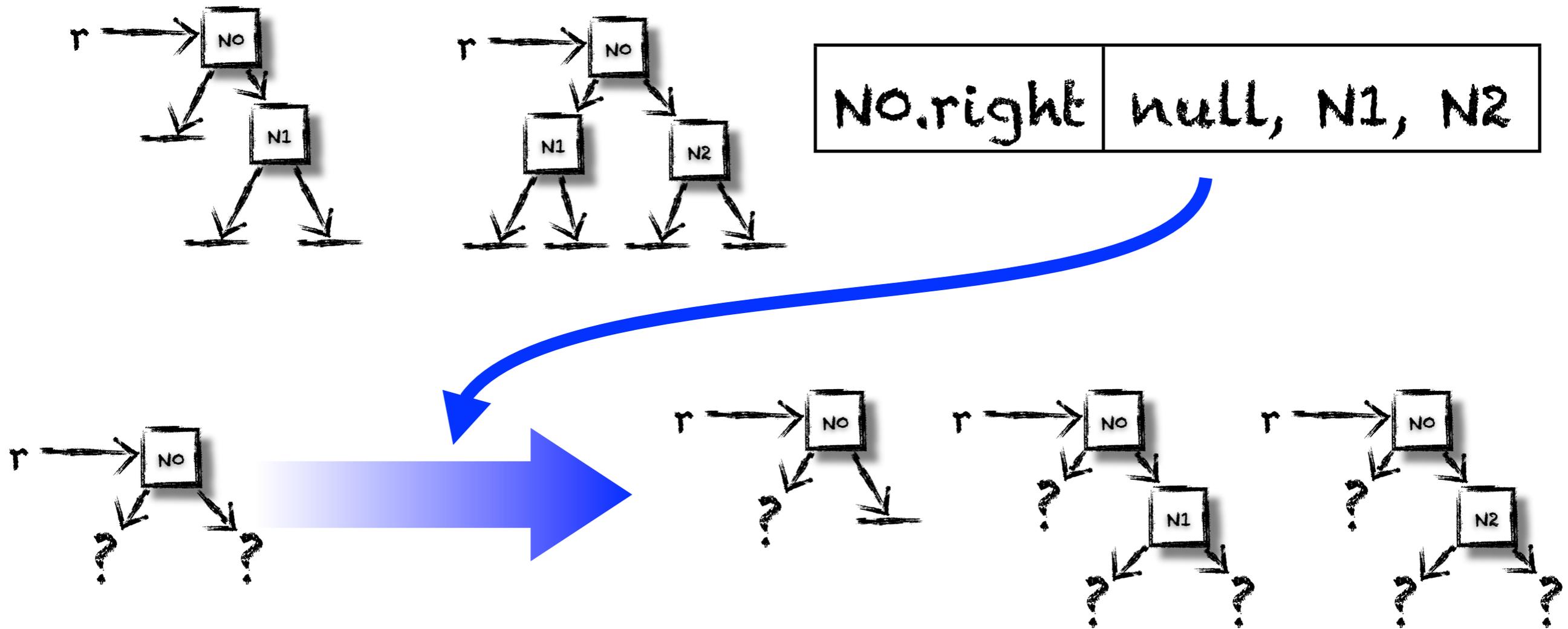
n	unique	LAZY	B-L
1	1	4	2
2	3	21	10
3	8	82	36
4	22	306	145
5	64	1140	668
6	196	4275	3554
7	625	16144	21265
8	2055	61332	140996

Binary Search Tree

LAZY VS BOUNDED-LAZY

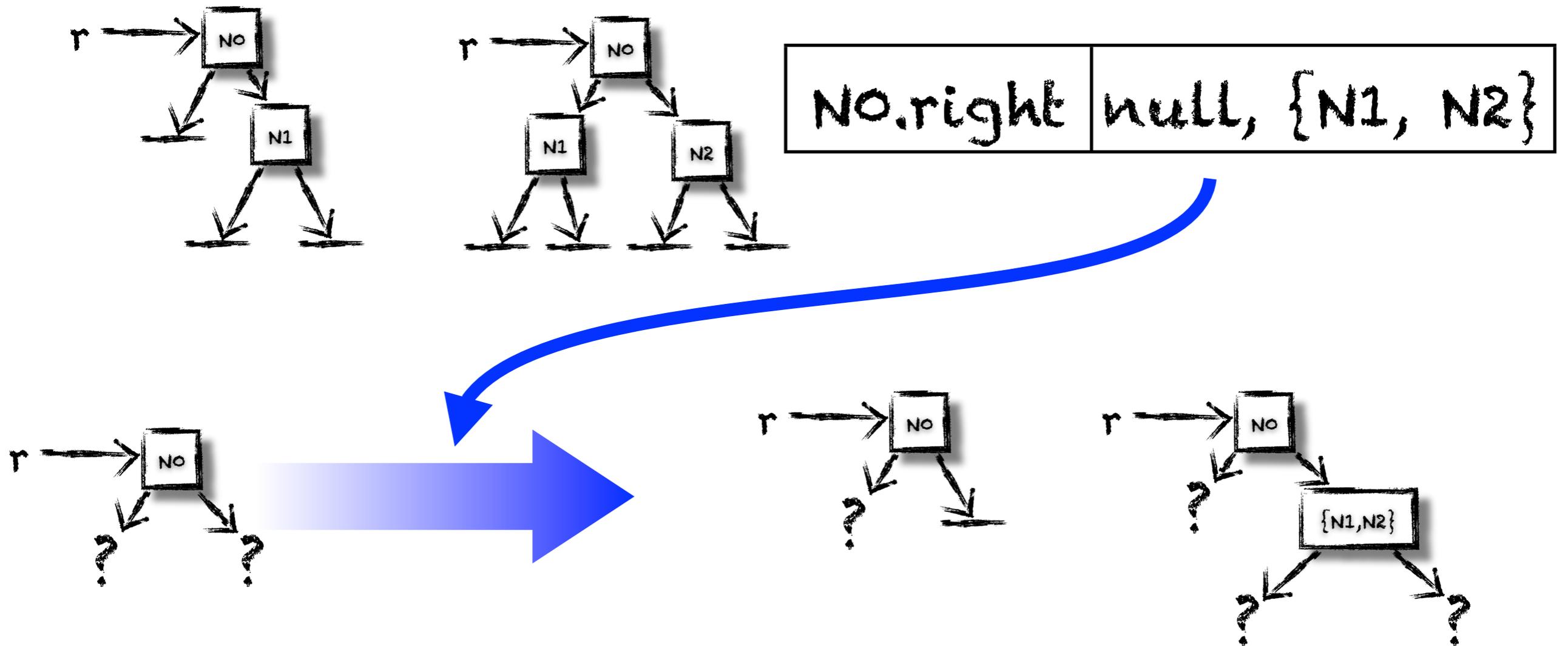


BOUNDED LAZY INITIALIZATION 2



While Bounded Lazy Initialization avoids (some) illegal structures, it unfortunately introduces duplicates. For large structures, this penalty overtakes the cost of illegal structures.

BOUNDED LAZY INITIALIZATION 2



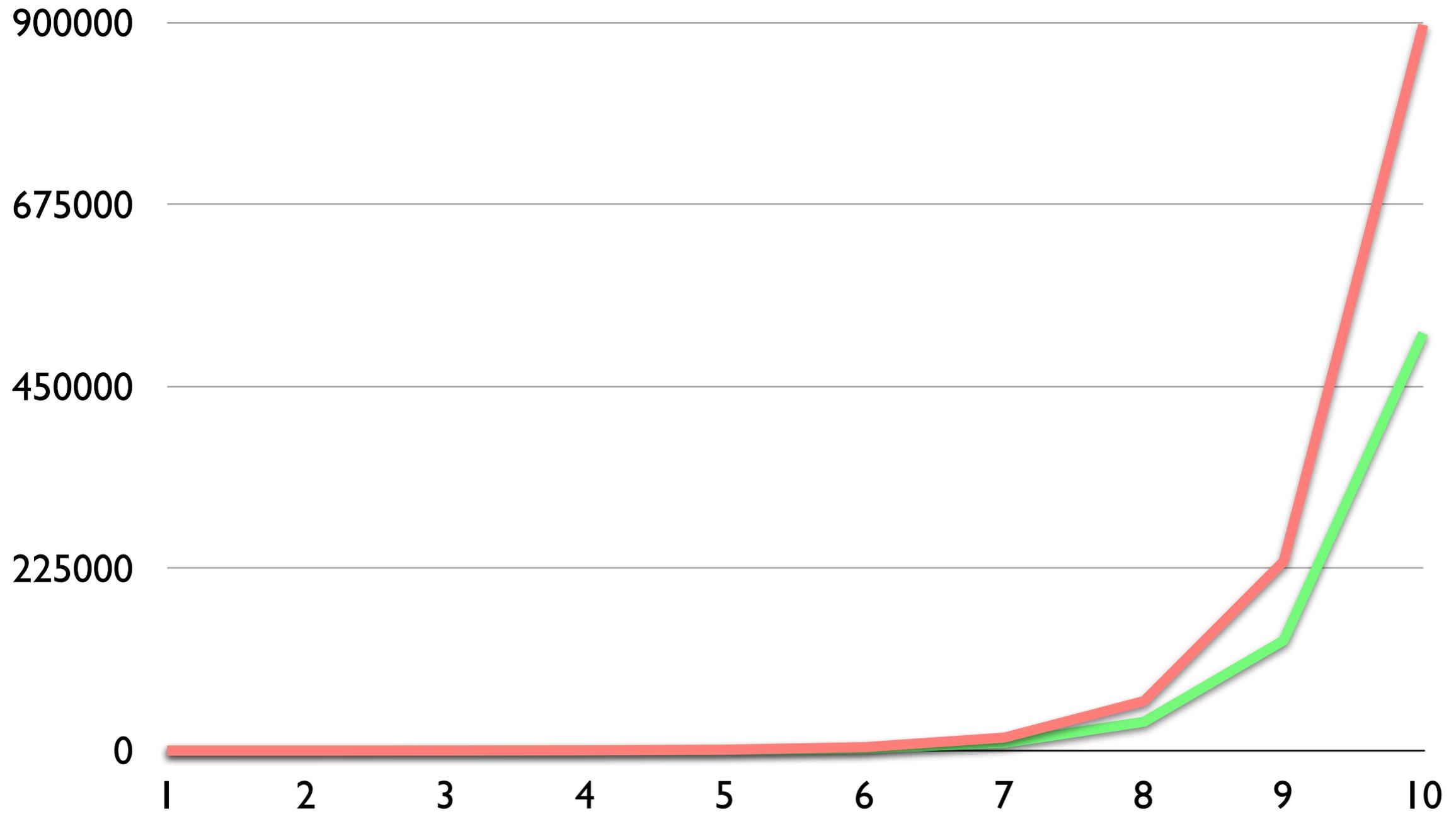
The symmetry-breaking algorithm we introduce in the paper addresses this problem by grouping some of the options together.

LAZY VS BOUNDED-LAZY2

n	unique	LAZY	B-L	B-L2
1	1	4	2	2
2	3	21	10	11
3	8	82	36	44
4	22	306	145	164
5	64	1140	668	639
6	196	4275	3554	2464
7	625	16144	21265	9604
8	2055	61332	140996	35695

Binary Search Tree

LAZY VS BOUNDED-LAZY2



LAZY VS BOUNDED-LAZY2

n	unique	LAZY	B-L2
1	1	2	1
10	10	74	19
100	100	5249	199

LinkedList

n	unique	LAZY	B-L2
1	2	4	2
2	4	27	15
3	7	110	21
4	15	409	101
5	29	1509	158
6	49	5610	883
7	84	21043	4715
8	148	79530	16146
9	270	302402	39583
10	518		149133

TreeSet

IN CONCLUSION

- We have shown that classic lazy initialization and bounded lazy initialization work well together.
 - Symmetry breaking is important to boost performance, but this only became apparent for larger structures.
 - This work symbolically executed the `repOK()` routine itself, to ensure that whole structures are generated.
- ➔ Additional ongoing work may produce further reductions in the number of structures that we explore.